Documentation on 48-hour project for candidacy

Timothy Jones, Ph.D.

January 26, 2014

Contents

1 Problem and summary of solution

1.1 Introduction

The following problem was posed:

Demonstration Project

We run over 50 servers currently and expect that number to grow rapidly. A configuration management tool allows us to automate provisioning and management of these servers, with the added benefit of being self-documenting.

Choose a configuration management tool and use it to provision a server with PHP, Apache, and MySQL. Then create a *very simple* webpage on the server that demonstrates use of these three systems (for example, a "hello world" PHP page that pulls out of the database and serves on apache).

You can spin up the server in a virtual machine (using something like VirtualBox or Linux KVM), or on a cloud-provider like Amazon AWS. Check out the Vagrant project as well - it might be useful.

Check your code into a public repository on github.com and send us a link - but be careful not to attach any secret keys or passwords!

Additional questions to think about: How will you get your simple webpage's code onto the machine? What would be different if you had to provision dozens of machines rather than just one? What did the configuration management tool make easy? What did it make difficult? Can you write any automated tests to ensure the configuration is correct?

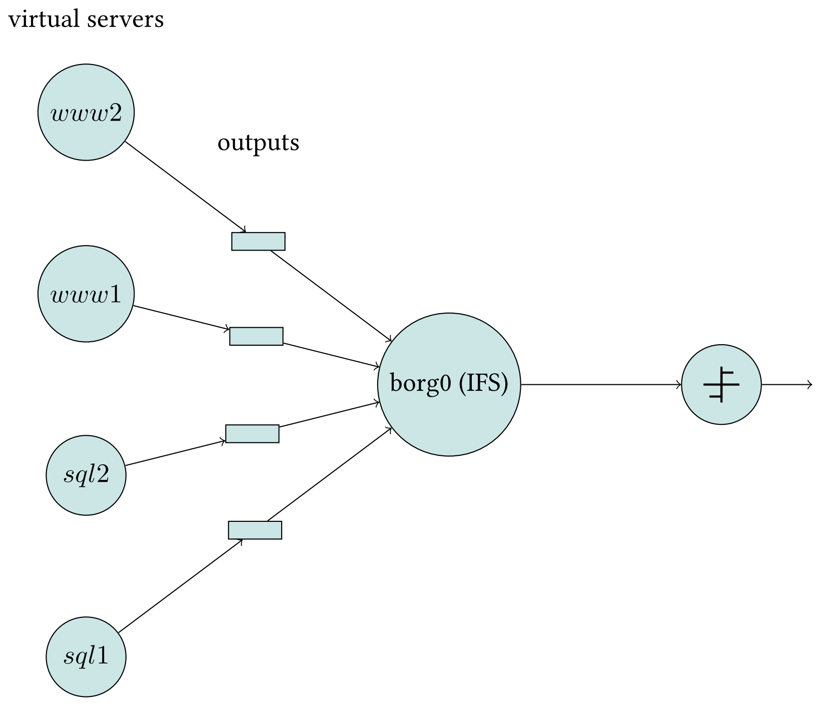
The best solutions to this problem will be easy for us to try. Additionally, the code and documentation will be thorough, thoughtful, and well presented. Use this as an opportunity to showcase your areas of expertise.

The solution to this problem uses virtual machines running on a Ubuntu 12.04 server. For security and practical reasons, it is preferable that the Apache/php server is separate from the MySQL server, and this solution makes this the case. Additionally, we demonstrate load-sharing by having two Apache/php servers host the same content and share content requests in a round-robin fashion. Additionally, these virtual servers are not directly exposed to the web and thus do not need to have their resources taxed by filtering/firewall activity.

2 Setting up basic infrastructure

2.1 Outline of network architecture

An unused server that I have permissions/access to for research purposes is being temporarily used to house this demonstration. It runs Ubuntu 12.04 LTS 64 bit. This server is called borg0.physics.drexel.edu and will hosts the virtual instances. The original plan was to host two redundant load-balanced Apache servers and two master-master coupled load-balanced MySQL servers, but for the sake of time, only one sql servers was used.



3 Building the system I

3.1 Choosing and setting up the correct configuration management tool (Vagrant and Chef)

In order to efficiently configure and manage a group of servers, a configuration management tool (CMT) is to be used. There are a range of open-source CMTs [9, 8] to choose from. For security reasons, it is important that the CMT supports **mutual authentication** and **encryption**.

Although this company does not currently use Windows servers, it would be best to keep that option open, and so we would also require that the CMT supports Linux/Unix (primary platform) as well as Windows. Mac OSX does not seem to have a strong server/database presence, and so we will not use Mac support as a criteria.

A final requirement would be that the CMT have **verify mode** which helps ensure the fidelity of upgrade.

These initial requirements narrowed down the list of potential CMTs to Chef, CFEngine, Puppet, Rundeck, and Salt. One can find many arguments for and against these CMTs online; however, a deciding factor is that AWS Opsworks [1] supports Chef-based scripts. As it was indicated that this company is inclined towards a continued and increasing use of Amazon AWS, it would be beneficial to use Chef as the CMT for the locally hosted virtual machines.

We require the creation of three virtual servers, two Apache/PHP servers, and one MySQL servers; these are hosted on an actual server that acts as a firewall and a load-balancing server, isolating the virtual servers with "sensitive" information from the internet and allowing those servers to dedicate their resources to their assigned tasks.

3.2 Internet facing server: the front yard

The load-balancing tasks will be given to the server which is hosting the virtual instances and which is filtering and directing external traffic for the virtual servers.

A basic installment of Ubuntu 12.04 LTS should be coupled with the packages **OpenSSH server**. The file `/etc/network/interfaces` should have the following form:

Listing 3.1: Interfaces

```
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet static
  pre-up iptables-restore < /etc/rc.firewall
```

```
   address 129.25.7.112
   netmask 255.255.128.0
   network 129.25.1.0
   broadcast 129.25.1.255
   gateway 129.25.1.253
   dns-nameservers 144.118.24.20 144.118.24.10

auto eth1
# The backyard
iface eth1 inet static
        address 192.168.2.100
        netmask 255.255.255.0
        broadcast 192.168.2.255
```

where the static ip address of the external and internal networks should be changed to reflect the proper values. Finally, we require the use of iptables[6] as a firewall and in order to perform network-address-translation so that the deployed network can communicate with the external network. While we ended up using vagrant which does this for the user, I'm including the details here because it would be most ideal to put the IFS behind a second server or hard-ware firewall whose job was exclusively to filter traffic. Such a task would not make sense for this small demo project.

The simplest way to deploy a firewall on Ubuntu systems is to use **UFW** (Uncomplicated Firewall), however, an iptables script can provide more transparent control of the firewall. Our file should take a form similar to:

Listing 3.2: Firewall

```
#!/bin/sh

ext=eth0
int=eth1

extip="129.25.7.112"

# Enable IP forwarding.
echo "1" > /proc/sys/net/ipv4/ip_forward

# Flush all rules.
/sbin/iptables -P INPUT ACCEPT
/sbin/iptables -F INPUT
/sbin/iptables -P OUTPUT ACCEPT
/sbin/iptables -F OUTPUT
/sbin/iptables -P FORWARD DROP
/sbin/iptables -F FORWARD
/sbin/iptables -t nat -F

#####
```

```
# Masquerade for internal network at int via ext.

/sbin/iptables -A INPUT -i $ext -m state --state ESTABLISHED,RELATED -j
    ↪ ACCEPT
/sbin/iptables -A FORWARD -i $ext -o $int -m state --state ESTABLISHED,
    ↪ RELATED -j ACCEPT
/sbin/iptables -A FORWARD -i $int -o $ext -j ACCEPT

# Use SNAT for static ext interface, MASQUERADE for DHCP ext interface.
if [ -n "$extip" ]; then
        /sbin/iptables -t nat -A POSTROUTING -o $ext -j SNAT --to-source $extip
else
        /sbin/iptables -t nat -A POSTROUTING -o $ext -j MASQUERADE
fi

# Let anything in through int, i.e. trust the deployed network
/sbin/iptables -A INPUT -i $int -j ACCEPT

# Permit ssh and http traffic
/sbin/iptables -A INPUT -i $ext --protocol tcp --dport ssh -j ACCEPT
/sbin/iptables -A INPUT -i $ext --protocol tcp --dport http -j ACCEPT
/sbin/iptables -A INPUT -i $ext --protocol tcp --dport https -j ACCEPT

# Ping requests help determine whether or not the server is connected to network
/sbin/iptables -A INPUT --protocol icmp --icmp-type echo-request -j ACCEPT

# Unlimited access to the loopback interface
/sbin/iptables -A INPUT -i lo -j ACCEPT
/sbin/iptables -A OUTPUT -o lo -j ACCEPT

/sbin/iptables -A INPUT -p all -s localhost -i eth0 -j DROP
# -Deny outside packets from internet whichclaim to be from your loopback
    ↪ interface

# REJECT (rather than DROP) any INPUTs that do not match one of the above rules
/sbin/iptables -A INPUT -i $ext -p tcp -j REJECT
/sbin/iptables -A INPUT -i $ext -p udp -j REJECT
```

which is saved to the file `/etc/rc.firewall`. We can ensure these iptables rules are always coupled with a connection to the web by adding the line

```
pre-up iptables-restore < /etc/rc.firewall
```

to the eth0 configuration in `/etc/network/interfaces` (see Listing 3.1), we can ensure that these rules are activated whenever eth0 is activated, i.e. whenever the IFS is online.

Finally, in order to short-circuit brute-force ssh attacks, it is useful to install fail2ban, which temporarily blacklists an ip address attempting to log in with an incorrect password three times in a row:

```
sudo apt-get install fail2ban
```

Load-balancing

Obvious for a demo such as this there is no need for load-balancing, but such a tool would be valuable for a scaled-up site. On Linux, the haproxy tool is commonly used and can be enabled on the IFS as follows:

```
sudo apt-get install haproxy
sudo nano /etc/default/haproxy # Change ENABLED=0 to ENABLED=1
sudo mv /etc/haproxy/haproxy.cfg{,.original}
```

By creating the file `/etc/haproxy/haproxy.cfg` with the following contents:

Listing 3.3: ”haproxy.cfg”

```

defaults
    log global
    mode http
    option httplog
    option dontlognull
    retries 3
    option redispatch
    timeout connect 5000
    timeout client 10000
    timeout server 10000

listen borg0.physics.drexel.edu 0.0.0.0:80
    mode http
    balance roundrobin
    option httpclose
    option forwardfor
    server lamp1 192.168.33.11:80 check
    server lamp2 192.168.33.12:80 check
```

Here we’ve used the round-robin balancing method which alternates request between the two web servers we will be using, though other methods exist, including “least connections”. Haproxy can also be used to load balance two or more mysql servers, although we haven’t used this capability for this demo. Another important feature here is that if one of the virtual web-servers were to crash, Haproxy would seamlessly route all of its traffic to the other web-server, and the users would not notice the difference.

4 Building the system II

4.1 Deployment network: the back yard

With our basic internet facing server deployed, it is now time to deploy the Apache/PHP/MySQL network. We do this on a second Ubuntu server which has been given a default installation of Ubuntu 12.04 LTS. Although this network is protected with the firewall implemented on the IFS and is *not* directly connected to the internet, it would be good practice to imagine a worst case scenario in which the IFS has been penetrated and an attacker has full access to the internal network (the back yard). Hence we would also install fail2ban on this network and implement a firewall that restricts traffic from each virtual instance on the network to its designated ports, e.g. Apache/PHP servers would only accept 80/443 traffic, and the MySQL servers would only accept 3306. The security outlines for the IFS serve as a template for making such restrictions, but we would elect to use Ubuntu's **UFW** since it's simple operation makes it easy to configure on an ensemble of servers using Chef. However, we do not implement these extra security features for this demo project as they would not be practical for a small implementation such as this.

4.2 Initiating Ruby, Chef, and Vagrant

Our first step is, of course, the installation of **Vagrant** and **Chef** [7, 5, 2] on the server which will host the virtual back yard. We can install Vagrant via the Ubuntu repository, but to install Chef we must first install Ruby–the language Chef is written in–which is best done via the Ruby Version Manager RVM [3]

```
sudo apt-get install vagrant
sudo apt-get install git-core curl
sudo -s
\curl -L https://get.rvm.io | bash -s stable --ruby=1.9.3
exit
source /usr/local/rvm/scripts/rvm
```

The first line, to install vagrant, will also install virtualbox and other packages necessary for vagrant to work. The Ruby version is chosen to best comply with the requirements of Chef (+1.9.3)[4] though it isn't the most current maintained version. Next we install chef via gem:

```
sudo gem install chef --no-ri --no-rdoc
gem install knife-solo --no-ri --no-rdoc
```

where the option `--no-ri` and `--no-rdoc` prevent the installation of extra documentation features that we don't need for this demo. Once Chef is installed, we can use tools provided by Opscode to organize our recipes:

```
cd; mkdir DEMOdata; cd DEMOdata
wget http://github.com/opscode/chef-repo/tarball/master
tar -zxvf master
mv opscode-chef-repo* chef-repo
ln -is ~/DEMOdata/chef-repo/cookbooks .
mkdir ~/.chef; echo "cookbook_path [ '/home/username/DEMO/chef-repo/
    ↪ cookbooks' ]" > .chef/knife.rb
```

We will have two main folders, one will be called DEMO which will store the Vagrantfile needed to configure the instances, as well as the shared webpages and databases, and another called DEMOdata that will be have recipes and may also contain sensitive information such as passwords. Anything in the DEMO folder will be seen on all instances under a mounted /vagrant folder, and so this separation is a security-driven decision. The above commands create a suggested file structure for storing our Chef-related files, and the last line directs the Chef command knife to our recipe directory. Knife is used to interface the recipe repository with the chef server and gathering recipes from online sources. We also linked the folder "cookbooks" to the DEMO directory as this is the default location that Vagrant uses to look for cookbooks.

4.3 Apache/PHP servers

Now we are ready to set up our two virtual Apache/PHP servers. We wish to have full control over what web-pages they serve, that is, we wish to find an automated way to update the webpages so that if we had, say, hundreds of redundant servers, we could very easily get them all serving the same web-page in an instant push rather than a time-consuming roll-out. Because all Vagrant instances will share what they find in the `~/DEMO` directory, we create a folder in that directory called `www` and populate it with a place-holder webpage for now:

```
cd ~/DEMO; mkdir www; echo test > ~/DEMO/www/index.html
```

We will add our demo software to this folder later, but first we need to install an Apache/PHP server and get it up and running and transparent to the real-world, and then add a second such server and have them share the work load and act as mutual backups in case one of them has a fault.

Our first task is to install the Chef recipe for `apache2`, `users`, and `chef-solo-search`. We store these in a separate folder called `DEMOdata` since they will contain some sensitive information such as passwords and we do **not** want to expose those files to the shared /vagrant directory that each instance will see:

```
cd ~/DEMOdata/cookbook
knife cookbook site download apache2
knife cookbook site download users
knife cookbook site download chef-solo-search
knife cookbook site download mysql
knife cookbook site download php
```

```
sudo knife cookbook site download php
knife cookbook site download sudo
git://github.com/fnichol/chef-openssh.git
sudo gem install ruby-shadow
tar *.tar.gz
```

Now we need to modify this recipe so that our root document folder for Apache is the shared `/vagrant/` folder which makes for easy deployment of changes to the webpage.

Anticipated Q&A

Q: If the two virtual servers share the same root-document folder, won't that partially defeat the purpose of having two servers as the IO operations on that folder would be the same as if there was only one server?

A: The Linux kernel stores frequently accessed files in RAM memory, checking only if the files have changed on disk but not actually accessing their contents on disk unless changes have been made. Thus the root documents will be copied into the memory of each server, rather than read directly from disk each time a page request comes in.

4.3.1 Pointing Apache root folder to shared folder

In order that our two Apache servers will be sharing the exact same web-site under a load-balancing scheme, we require that Chef configures the Apache installs to point to that directory rather than the default `/var/www`. We modify the file:

`~/DEMO/cookbooks/apache2/attributes/default.rb`

on line 55 (which pertains to Ubuntu variations of Linux) to read:

```
default['apache']['docroot_dir'] = '/vagrant/www'
```

4.3.2 Installing a user

This section is not necessary for the demonstration, but it can sometimes be preferable to have a user other than “vagrant”. In the same folder as “cookbooks”, run the following command:

```
mkdir -p databags/users
%mkdir -p databags/ssh_known_hosts
```

create a password for the user by taking the output of this command:

```
openssl passwd -1 "whateverpasswd"
```

and putting it into a file you will create in databags/users called sysadmin.json

Listing 4.1: "sysadmin.json"

```
{
  "groups": [
    "sysadmin"
  ],
  "comment": "System Administrator",
  "password": "$1$vRQiX4XZ$hhTwHNMNUpY2GJcjxWW8c2",
  "id": "sysadmin",
  "shell": "/bin/bash",
  "email": "tdj82@drexel.edu"
}
```

At this point our web-server is ready to be initiated with **vagrant**, but before taking that final step, we want to set up a configuration for the Mysql server.

4.4 MySQL servers

4.4.1 Creating a demonstration database

We first need to create a demonstration database that we wish to load into the MySQL server. We do this by creating a MySQL script and running it on a development system that has mysql-server installed:

Listing 4.2: "demodb_script.sql"

```
CREATE DATABASE demo;
GRANT USAGE ON *.* TO demouser@localhost IDENTIFIED BY 'xxxxxxxx';
GRANT ALL PRIVILEGES ON demo.* TO demouser;
USE demo;
CREATE TABLE products
 (id INTEGER NOT NULL,
 name VARCHAR(25) NOT NULL,
 manufacturer VARCHAR(25) NOT NULL,
 quantity_in_house INTEGER NOT NULL,
 minimum_to_stock INTEGER NOT NULL,
 maximum_to_stock INTEGER NOT NULL,
 cost_per_unit FLOAT NOT NULL,
 retail_per_unit FLOAT NOT NULL,
 not_discontinued ENUM('T','F') NOT NULL);
INSERT INTO products (id, name, manufacturer, quantity_in_house,
    ↪ minimum_to_stock,maximum_to_stock,cost_per_unit,retail_per_unit,
    ↪ not_discontinued) VALUES(1000,'iTab','Plum Inc.', 432, 350, 1200, 335.25,
    ↪ 499.99, 'T');
INSERT INTO products (id, name, manufacturer, quantity_in_house,
    ↪ minimum_to_stock,maximum_to_stock,cost_per_unit,retail_per_unit,
    ↪ not_discontinued) VALUES(1001,'Beats by Dru','BD Inc.', 132, 230, 500,
    ↪ 55.10, 99.99, 'T');
INSERT INTO products (id, name, manufacturer, quantity_in_house,
    ↪ minimum_to_stock,maximum_to_stock,cost_per_unit,retail_per_unit,
    ↪ not_discontinued) VALUES(1002,'Blue Soda','WW Industries', 32, 50, 200,
    ↪ 1.00, 3.99, 'F');
INSERT INTO products (id, name, manufacturer, quantity_in_house,
    ↪ minimum_to_stock,maximum_to_stock,cost_per_unit,retail_per_unit,
    ↪ not_discontinued) VALUES(1003,'Nebula S IV','Dangson Inc.', 552, 200, 500,
    ↪  280.00, 399.99, 'T');
INSERT INTO products (id, name, manufacturer, quantity_in_house,
    ↪ minimum_to_stock,maximum_to_stock,cost_per_unit,retail_per_unit,
    ↪ not_discontinued) VALUES(1004,'eWatch','App-el Inc.', 2, 50, 100, 135.25,
    ↪ 249.99, 'T');
```

We run the script to create the new database:

```
mysql -uroot -pxxxxxx < demodb_script.sql
```

Then we dump this database and copy the dumpfile over to the vagrant shared directory under a folder called sql:

```
sudo mysqldump -uroot -pxxxxx --opt demo > demo.sql
```

In the previous section, we have already downloaded the recipes necessary for the mysql server. We now need to create our own recipe to have this server load . We use knife to create a template for the cookbook (in the main vagrant directory):

```
sudo knife cookbook create demoload -o cookbooks/ -r md
```

In order to provide us some flexibility, we will have this recipe run a bash script in the vagrant main directory once the mysql server is fully loaded. This allows us to add additional databases if necessary without having to edit the recipe.

In the folder `/cookbooks/demoload/recipes`, we create the default.rb file with contents:

```
execute "databaseup" do
    command "/vagrant/bootstrap.sh"
    user "root"
end
```

and we create the bootstrap.sh file to have the following contents:

Listing 4.3: ”bootstrap.sh”

```
#!/usr/bin/env bash
if [ ! -f /var/log/databasesetup ];
then
    echo "CREATE USER 'demouser'@'localhost' IDENTIFIED BY 'xxxxxx'" |
        ↪ mysql -uroot -pxxxxxxx
    echo "CREATE USER 'demouser'@'%' IDENTIFIED BY 'xxxxxx'" | mysql -uroot
        ↪  -pxxxxxxxx
    echo "CREATE DATABASE demo" | mysql -uroot -pxxxxxxxxxx
    echo "GRANT ALL PRIVILEGES on *.* TO demouser@'%' IDENTIFIED BY '
        ↪ xxxxxx' with grant option" | mysql -uroot -pxxxxxxxxxx;
    #echo "GRANT ALL ON demo.* TO 'demouser'@'localhost'" | mysql -uroot -
        ↪ pxxxxxxxxxx
    #echo "GRANT ALL ON demo.* TO 'demouser'@'%'" | mysql -uroot -
        ↪ pxxxxxxxx
    echo "flush privileges" | mysql -uroot -pxxxxxxxxx

    touch /var/log/databasesetup

    if [ -f /vagrant/sql/demo.sql ];
    then
        mysql -uroot -ponlyneedthisforsolo demo < /vagrant/sql/demo.sql
       /etc/init.d/mysql restart
   fi
fi
```

This will take the demo database we created previously and put in the vagrant/sql folder and restore it on the deployed sql server.

4.5 Deployment

We are now ready to deploy the system (two www servers and one sql server). We create the following Vagrantfile:

Listing 4.4: Vagrantfile

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

Vagrant::Config.run do |config|

  config.vm.box = "precise64"
  config.vm.box_url = "http://files.vagrantup.com/precise64.box"

  config.vm.define "sql1" do |sql1|
        sql1.vm.network :hostonly, "192.168.33.13"
        sql1.vm.provision :chef_solo do |chef3|
            chef3.data_bags_path = "../DEMOdata/databags"
            chef3.cookbooks_path = "../DEMOdata/cookbooks"
            chef3.add_recipe "chef-solo-search"
            chef3.add_recipe "apt"
            chef3.add_recipe "users::sysadmins"
            chef3.add_recipe "sudo"
            chef3.add_recipe "openssl"
            chef3.add_recipe "mysql"
            chef3.add_recipe "mysql::server"
            chef3.add_recipe "demoload"
            chef3.add_recipe "openssh"
            chef3.json = {
                        :apache=> { :default_site_enabled => true },
                        :authorization=> {
                                    :sudo=> {
                                    :passwordless=> true,
                                    :users=> ["sysadmin"],
                                    :groups=> ["admin", "sudo"]
                                       }
                        },
                         :openssh=> {
                                  :permit_root_login=> "no",
                                  :password_authentication=> "yes"
                                  },

                         :mysql=> {
                                  :server_root_password=> "xxxxxx",
                                  :server_repl_password=> "xxxxxx",
                                  :server_debian_password=> "xxxxx",
                                  :bind_address=>"192.168.33.13"
                          }

               }

     end
end

config.vm.define "www1" do |www1|
         www1.vm.network :hostonly, "192.168.33.11"
         www1.vm.provision :chef_solo do |chef1|
            chef1.data_bags_path = "../DEMOdata/databags"
            chef1.cookbooks_path = "../DEMOdata/cookbooks"
            chef1.add_recipe "chef-solo-search"
            chef1.add_recipe "apt"
            chef1.add_recipe "users::sysadmins"
            chef1.add_recipe "sudo"
            chef1.add_recipe "apache2"
            chef1.add_recipe "apache2::mod_php5"
            chef1.add_recipe "php::module_mysql"
            chef1.add_recipe "openssh"
            chef1.json = {
                        :apache=> { :default_site_enabled => true },
                        :authorization=> {
                                    :sudo=> {
                                    :passwordless=> true,
                                    :users=> ["sysadmin"],
                                    :groups=> ["admin", "sudo"]
                                      }
                        },
                         :openssh=> {
                                  :permit_root_login=> "no",
                                  :password_authentication=> "yes"
                                  },

                          }



       end
   end

   config.vm.define "www2" do |www2|
          www2.vm.network :hostonly, "192.168.33.12"
          www2.vm.provision :chef_solo do |chef2|
             chef2.data_bags_path = "../DEMOdata/databags"
             chef2.cookbooks_path = "../DEMOdata/cookbooks"
             chef2.add_recipe "chef-solo-search"
             chef2.add_recipe "apt"
             chef2.add_recipe "users::sysadmins"
             chef2.add_recipe "sudo"
             chef2.add_recipe "apache2"
             chef2.add_recipe "apache2::mod_php5"
             chef2.add_recipe "php::module_mysql"
             chef2.add_recipe "openssh"
             chef2.json = {
                        :apache=> { :default_site_enabled => true },
                        :authorization=> {
                                    :sudo=> {
                                    :passwordless=> true,
                                    :users=> ["sysadmin"],
                                    :groups=> ["admin", "sudo"]
                                      }
```

```
                              },
                               :openssh=> {
                                          :permit_root_login=> "no",
                                          :password_authentication=> "yes"
                                          },

                                }

      end
   end
end
```

4.5.1 sql1

This installs three machines running Ubuntu 12.04 LTS (64bit). The first is our sql machine, called sql1 and given the ip address 192.168.33.13. The combinations of recipes for this instance install a new user called "sysadmin" with a pre-defined password as well as installing mysql-server (the other recipes satisfy necessary dependencies). It also installs the demoload recipe we wrote in order to import the demo database when the instance is created. Because this is all automated with Vagrant and Chef, it would be easy to deploy this and any number of database to dozens of such virtual instances.

Although openssh is installed by default with Ubuntu 12.04, the openssh recipe allows us to turn off root login automatically, as seen in the json section below the recipe entries. The authorization json section sets up the sysadmin user we created. The mysql json section is necessary as we are running chef-solo instead of chef-server, and so passwords need to be entered here. We change the bind address here in order for the mysql server to point towards the common internal network (192.168.33.*).

4.5.2 www1,www2

The two Apache servers are mapped to www1 with ip address 192.168.33.11 and www2 with ip address 192.168.33.12. Instead of the mysql recipe as for sql1, we install the apache2 recipe, being sure to include the necessary php5 module, as well as the php recipe so that we can include the mysql php recipe. The json entries here are similar to those for the sql1 server.

4.5.3 Up and running

At last, to get this system up and running, we run:

```
sudo vagrant up
```

We can break down the entire system with the command:

```
sudo vagrant destroy
```

We can raise a single instance with the command, for example,:

```
sudo vagrant up www1
```

Or we can breakdown a single instance with the command, for example:

```
sudo vagrant destroy sql1
```

After running `sudo vagrant up`, the entire system is up and running within about ten to fifteen minutes. This is an older server, so that number would probably be much faster on a newer machine. Additionally, once one is happy with the behavior of the system, a new virtual box can be made from each instance so that future initiation would go much more quickly, not require access to a remote virtual box server, and require very simple Vagrantfile configurations.

4.6 Note on Amazon AWS

Amazon's AWS Opsworks [1] presents an incredibly stream-lined way to set up app infrastructure and deploy applications. It supports Chef scripts and so refinement of configurations is possible.

Vagrant can also be tied in with Amazon AWS so that the instances created in this demonstration can be deployed on the Amazon cloud without configuration changes (except for changes needed in the Vagrantfile to point to Amazon instead of the Ubuntu virtual box).

5 Live example and testing the system

To complete this demonstration, we created a simple php file that pulls the demo database from the database server and creates dynamic html content served via Apache2. Haproxy interacts with both web servers set up in this networking, sending them any web requests in a round-robbin fashion. Thus a user might point their browser at the internet facing server and think it appears to host a web-page, though it does not–the actual hosts are www1 and www2 with a third database host sql1 keeping the mysql server separate from the apache server.

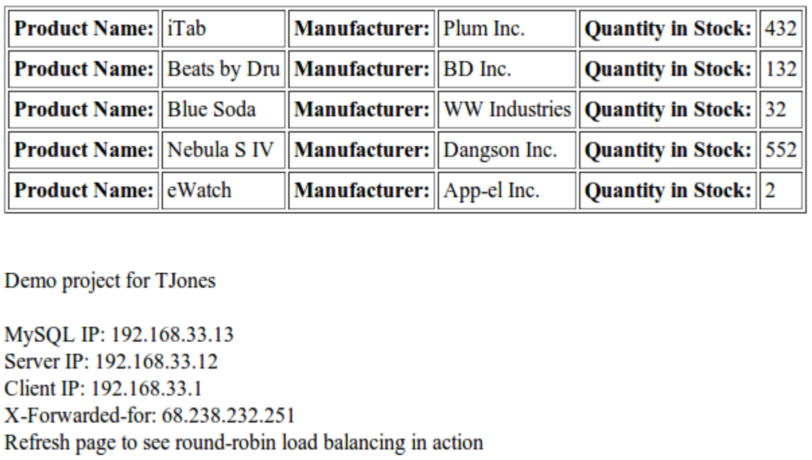
The php file used to do this is called `test.php`:

Listing 5.1: "test.php"

```
 <?php
 mysql_connect("192.168.33.13", "demouser", "xxxxxxxx") or die(mysql_error());
 mysql_select_db("demo") or die(mysql_error());
 $data = mysql_query("SELECT * FROM products")
 or die(mysql_error());
 Print "<table border cellpadding=3>";
 while($info = mysql_fetch_array( $data ))
 {
 Print "<tr>";
 Print "<th>Product Name:</th> <td>".$info['name'] . "</td> ";
Print "<th>Manufacturur:</th> <td>".$info['manufacturer'] . " </td>";
 Print "<th>Quantity in Stock:</th> <td>".$info['quantity_in_house'] . " </td></tr
     ↪ >";
 }
 Print "</table>";
 echo "<br><br>Demo project for TJones<br><br>MySQL IP: 192.168.33.13 <br>";
 echo "Server IP: ".$_SERVER['SERVER_ADDR'];
 echo "<br>";
 echo "\nClient IP: ".$_SERVER['REMOTE_ADDR'];
 echo "<br>";
 echo "\nX-Forwarded-for: ".$_SERVER['HTTP_X_FORWARDED_FOR'];
 echo "<br>";
 echo "Refresh page to see round-robin load balancing in action"

 ?>
```

Pointing the web-browser to `http://borg0.physics.drexel.edu/file.php` results in the following:

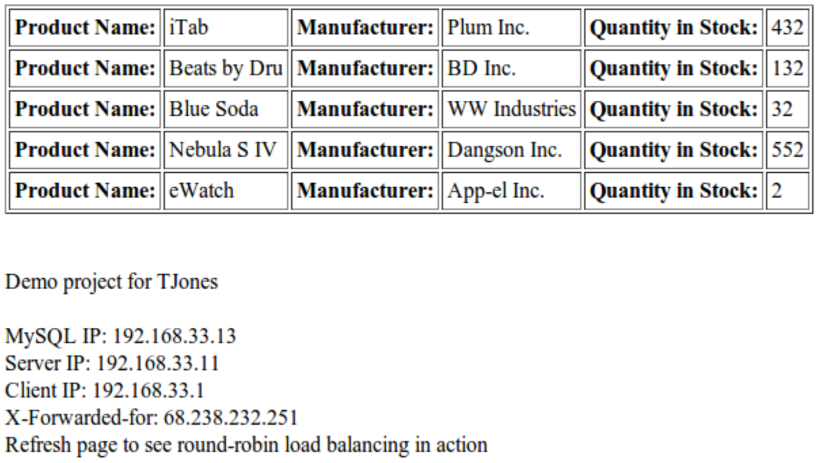


Product Name:	iTab	**Manufacturer:**	Plum Inc.	**Quantity in Stock:**	432
Product Name:	Beats by Dru	**Manufacturer:**	BD Inc.	**Quantity in Stock:**	132
Product Name:	Blue Soda	**Manufacturer:**	WW Industries	**Quantity in Stock:**	32
Product Name:	Nebula S IV	**Manufacturer:**	Dangson Inc.	**Quantity in Stock:**	552
Product Name:	eWatch	**Manufacturer:**	App-el Inc.	**Quantity in Stock:**	2

Demo project for TJones

MySQL IP: 192.168.33.13
Server IP: 192.168.33.12
Client IP: 192.168.33.1
X-Forwarded-for: 68.238.232.251
Refresh page to see round-robin load balancing in action

A refresh will yield:



Product Name:	iTab	**Manufacturer:**	Plum Inc.	**Quantity in Stock:**	432
Product Name:	Beats by Dru	**Manufacturer:**	BD Inc.	**Quantity in Stock:**	132
Product Name:	Blue Soda	**Manufacturer:**	WW Industries	**Quantity in Stock:**	32
Product Name:	Nebula S IV	**Manufacturer:**	Dangson Inc.	**Quantity in Stock:**	552
Product Name:	eWatch	**Manufacturer:**	App-el Inc.	**Quantity in Stock:**	2

Demo project for TJones

MySQL IP: 192.168.33.13
Server IP: 192.168.33.11
Client IP: 192.168.33.1
X-Forwarded-for: 68.238.232.251
Refresh page to see round-robin load balancing in action

where we can note that the Server IP address has changed indicating that indeed the web-traffic is being load-balanced.

Another simple script is created to give the user a command line option to see the load-balancing in action:

```
curl http://borg0.physics.drexel.edu/file.php
```

repeating this command will demonstrate the variation in the servers used to serve the webpage. To specifically address questions given in the assigned problem that haven't been addressed in any of the previous discussions:

- scp or even git could be used to get the web-content onto the internet-facing server which then shares it with the deployed web-servers.

- Vagrant makes it very easy to deploy dozens of machines. Now that the groundwork has been laid in terms of finding the right combination of recipes and settings, we can add many more instances very easily simply by editing Vagrantfile. Haproxy makes load-balancing very easy for Apache, and it is capable, although we didn't explore this option, to also load-balance mysql servers. In such a way, this simple demo could be scaled up, and one could even imagine scripts that could automate this scaling based on overall loads using Vagrant as the main tool.

- The configuration manager Chef saves time by automatically deploying configurations. Though is may be quicker to an experienced administrator to set up one computer as needed and then clone it, Chef makes that task systematic and salable in a way that simple cloning would not provide.

- We created a test to ensure the load-balancing was correct. As this was a 48-hour demo projects, we left much off the table, such as security (we can tighten the demouser's permissions on the mysql server), and so on.

This demo has shown, on a small scale, a systematic way to configure and deploy virtual servers to serve a Apache2/PHP/MySQL content. It demonstrated load-sharing capability between multiple Apache servers. Above all else, this demo could be very easily scaled up to a larger infrastructure and be able to grow/shrink dynamically as needed. For example, in a slow period, a number of the Apache servers could be destroyed. Haproxy would see them missing and route traffic to the remaining servers without any loss of service. If web traffic were to spike, new instances could be created and Haproxy could immediately start to share the work load with these new instances.

Bibliography

[1] Amazon.com. *AWS Opsworks*. [Online; accessed 23-January-2014]. 2014. URL: `http://aws.amazon.com/opsworks`.

[2] *Getting Started with Chef*. [Online; accessed 23-January-2014]. 2014. URL: `http://gettingstartedwithchef.com`.

[3] Digital Ocean. *How To Install Chef and Ruby with RVM on a Ubuntu VPS*. [Online; accessed 23-January-2014]. 2014. URL: `https://www.digitalocean.com/community/articles/how-to-install-chef-and-ruby-with-rvm-on-a-ubuntu-vps`.

[4] Opscode. *Chef Readme*. [Online; accessed 23-January-2014]. 2014. URL: `https://github.com/opscode/chef/blob/master/README.md`.

[5] OpsCode. *Chef Wiki*. [Online; accessed 23-January-2014]. 2014. URL: `https://wiki.opscode.com/display/chef/Home`.

[6] Ubuntu.com. *Iptables HowTo*. [Online; accessed 23-January-2014]. 2014. URL: `https://help.ubuntu.com/community/IptablesHowTo`.

[7] Vagrant. *Vagrant Docs*. [Online; accessed 23-January-2014]. 2014. URL: `http://docs.vagrantup.com/v2`.

[8] Peter Wayner. *Puppet or Chef: The configuration management dilemma*. [Online; accessed 23-January-2014]. Mar. 2013. URL: `http://www.infoworld.com/d/data-center/puppet-or-chef-the-configuration-management-dilemma-215279`.

[9] Wikipedia. *Comparison of open-source configuration management software — Wikipedia, The Free Encyclopedia*. [Online; accessed 23-January-2014]. 2014. URL: `http://en.wikipedia.org/wiki/Comparison_of_open-source_configuration_management_software`.

INDEX